实验8 实现运算及传送指令的CPU设计实验

一、实验目的

- 掌握RISC-V的R型和I型运算类指令、U型传送类指令的数据通路设计，掌握指令流和数据流的控制方法；
- 掌握完整的多周期CPU的系统结构，具备连接各模块构建整机的能力；
- 掌握基于有限状态机，实现控制单元的设计方法；
- 实现RISC-V的R型和I型运算类指令、U型传送类指令的功能。

二、实验内容与原理

- **实验内容：**
  - 以实现目标指令集为前提，首先将前述实验实现的各个逻辑模块连接，**构成RISC-V整机**；
  - 然后设计**二级译码及控制单元**，控制各个部件有序工作，实现目标指令集的功能；
  - 最后，**编写测试程序**，检验CPU是否能正确执行基于目标指令集的程序。

二、实验内容与原理

- **实验内容：实现R型的运算类指令10条、I型的运算类指令9条和U型指令lui**

1. 目标指令集
2. R型指令的数据通路与执行过程
3. I型指令的数据通路与执行过程
4. U型lui指令的数据通路与执行过程
5. 指令二级译码
6. 控制单元CU
7. 测试程序

1、目标指令集

■ R型指令格式、编码及功能

位数	7位	5位	5位	3位	5位	7位	指令功能
R型指令字段	funct7	rs2	rs1	funct3	rd	opcode	
汇编指令	机器指令编码						
add rd,rs1,rs2	0000000	rs2	rs1	000	rd	0110011	rs1+rs2→rd（加法）
sub rd,rs1,rs2	0100000	rs2	rs1	000	rd	0110011	rs1-rs2→rd（减法）
sll rd,rs1,rs2	0000000	rs2	rs1	001	rd	0110011	rs1<<rs2→rd（逻辑左移）
slt rd,rs1,rs2	0000000	rs2	rs1	010	rd	0110011	if(rs1<rs2) 1→rd else 0→rd（有符号数比较小于置数）
sltu rd,rs1,rs2	0000000	rs2	rs1	011	rd	0110011	if(rs1<rs2) 1→rd else 0→rd（无符号数比较小于置数）
xor rd,rs1,rs2	0000000	rs2	rs1	100	rd	0110011	rs1⊕rs2→rd（异或）
srl rd,rs1,rs2	0000000	rs2	rs1	101	rd	0110011	rs1>>rs2→rd（逻辑右移）
sra rd,rs1,rs2	0100000	rs2	rs1	101	rd	0110011	rs1>>>rs2→rd（算术右移）
or rd,rs1,rs2	0000000	rs2	rs1	110	rd	0110011	rs1 | rs2→rd（或运算）
and rd,rs1,rs2	0000000	rs2	rs1	111	rd	0110011	rs1 & rs2→rd（与运算）

1、目标指令集

- R型指令的共同特征：
  - 操作码字段：OP=0110011b
  - 指令功能：由功能码字段funct3指出
  - R型指令：三操作数指令
    - 源操作数：两个，rs1、rs2字段指定，寄存器(直接)寻址
    - 目的操作数：rd字段指定，寄存器(直接)寻址
  - 功能描述：rs1 (op) rs2→rd

1、目标指令集

■ I型指令格式、编码及功能

位数	12位		5位	3位	5位	7位	指令功能
I型指令字段	imm12		rs1	funct3	rd	opcode	
汇编指令	机器指令编码						
addi rd,rs1,imm12	imm[11:0]		rs1	000	rd	0010011	rs1+SE32(imm12)→rd（加法）
slti rd,rs1,imm12	imm[11:0]		rs1	010	rd	0010011	if(rs1<SE32(imm12)) 1→rd else 0→rd（有符号数比较小于置数）
sltiu rd,rs1,imm12	imm[11:0]		rs1	011	rd	0010011	if(rs1<SE32(imm12)) 1→rd else 0→rd（无符号数比较小于置数）
xori rd,rs1,imm12	imm[11:0]		rs1	100	rd	0010011	rs1⊕SE32(imm12)→rd（异或运算）
ori rd,rs1,imm12	imm[11:0]		rs1	110	rd	0010011	rs1 | SE32(imm12)→rd（或运算）
andi rd,rs1,imm12	imm[11:0]		rs1	111	rd	0010011	rs1 & SE32(imm12)→rd（与运算）
slli rd,rs1,shamt	0000000	shamt	rs1	001	rd	0010011	rs1<<UE32(shamt)→rd（逻辑左移）
srli rd,rs1,shamt	0000000	shamt	rs1	101	rd	0010011	rs1>> UE32(shamt)→rd（逻辑右移）
srai rd,rs1,shamt	0100000	shamt	rs1	101	rd	0010011	rs1>>> UE32(shamt)→rd（算术右移）

1、目标指令集

- I型指令的共同特征：
  - 操作码字段：OP=0010011b
  - 指令功能：由功能码字段funct3指出
  - I型指令：三操作数指令
    - 源操作数：两个，rs1寄存器（直接寻址），立即数imm32
    - 目的操作数：rd字段指定，寄存器(直接)寻址
  - 功能描述：rs1 (op) imm32→rd

1、目标指令集

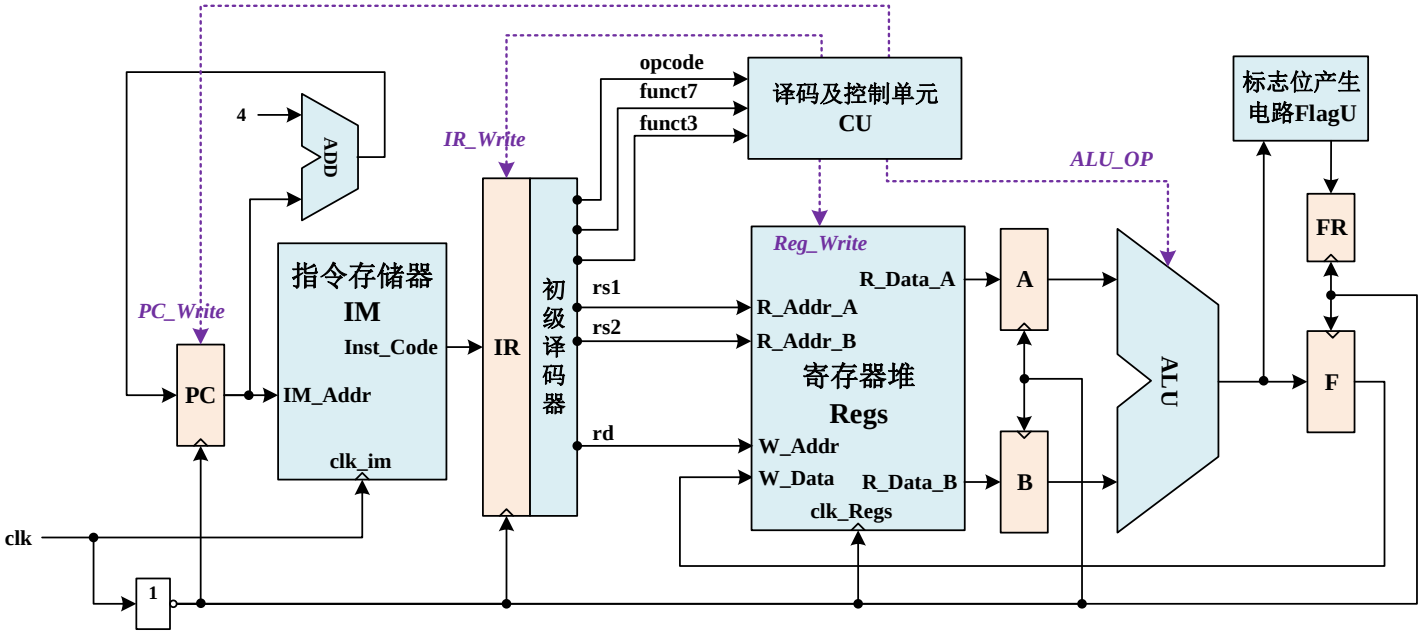
U型指令lui的格式、编码及功能

位数	20位	5位	7位	指令功能
U型指令字段	imm[31:12]	rd	opcode	
汇编指令	机器指令编码			
lui rd,imm20	imm[31:12]	rd	0110111	{imm20,12{0}}→rd（装立即数高20位）

- U型指令格式：大立即数格式
- 立即数：20位
- 功能：低位添加12个0后，装入rd寄存器
- lui指令+addi指令：可完成32位任意立即数的装数

2、R型指令的数据通路与执行过程

■ 1）R型指令的数据通路



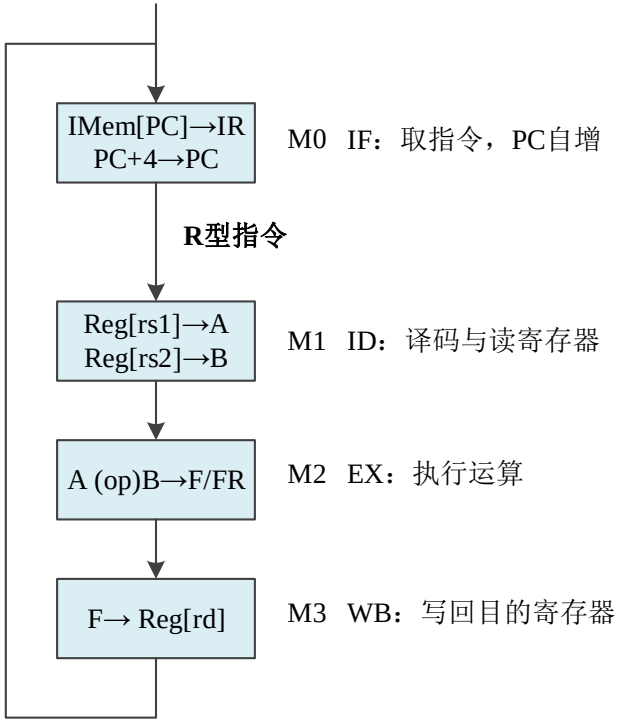
2、R型指令的数据通路与执行过程

1）R型指令的数据通路

- 从指令存储器取出的指令打入IR，进行初级译码
- 分解出的源寄存器rs1、rs2直接与寄存器堆的两个读端口A和B连接
- 将目的寄存器rd字段与寄存器堆的写端口地址相联
- 寄存器读出的A口数据和B口数据打入A和B暂存器后，A/B暂存器的输出与ALU的输入端连接
- ALU计算后的结果存入暂存器F，F的输出则送入寄存器堆的写数据端口

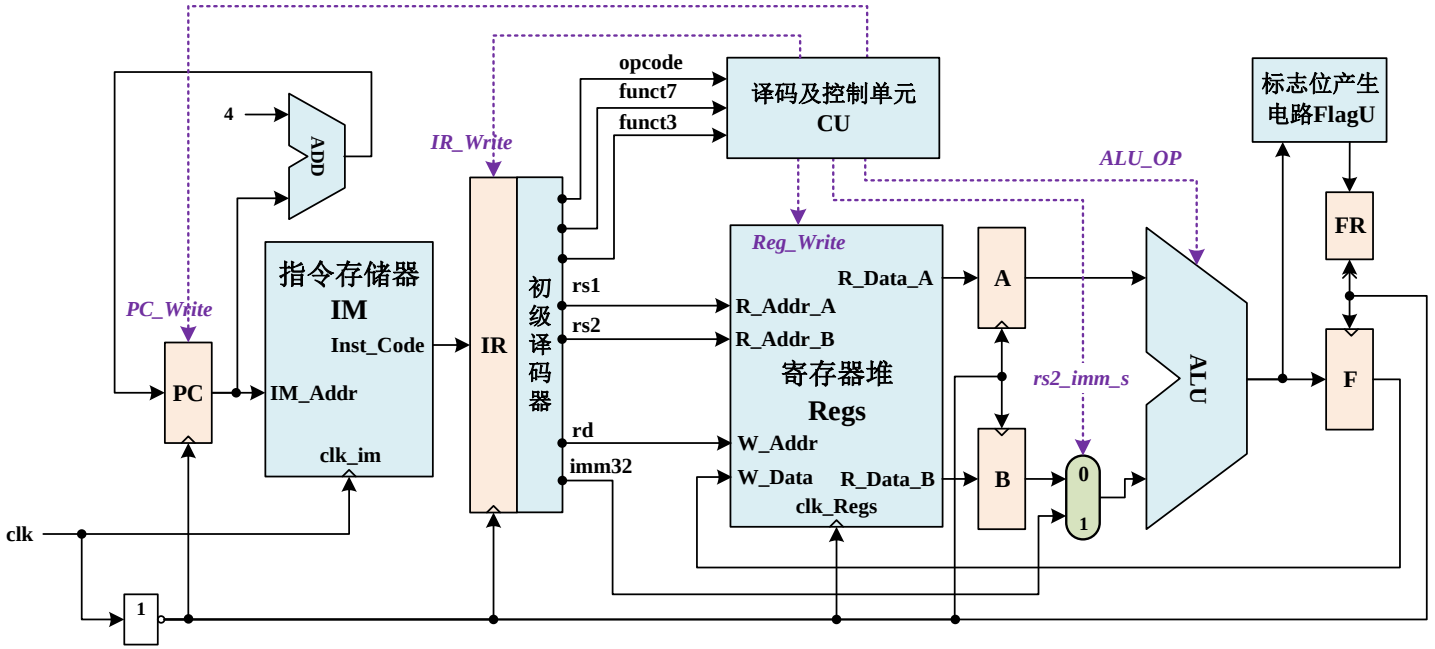
2、R型指令的数据通路与执行过程

■ 2）R型指令的执行过程



3、I型指令的数据通路与执行过程

■ 1）I型指令的数据通路



3、I型指令的数据通路与执行过程

1）I型指令的数据通路

- ALU运算的B输入端：添加一个32位的二选一数据选择器，由rs1_imm_s信号来选择：
  - R型指令：rs1_imm_s=0，选择从暂存器B提供ALU运算的B数据（ALU_B）
  - I型指令：rs1_imm_s=1，选择由初级译码器的立即数拼接与扩展器模块输出的imm32提供ALU_B

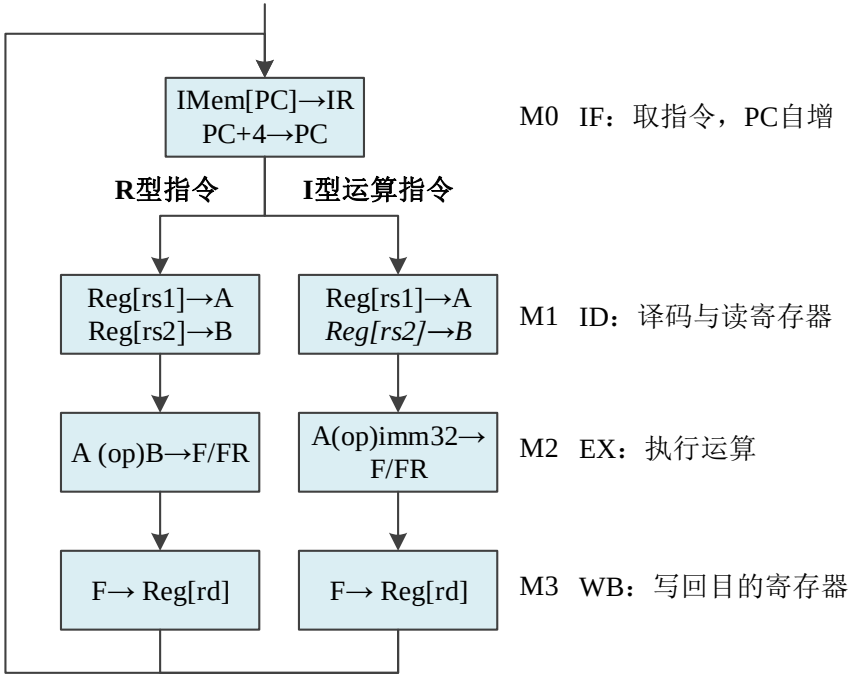
ALU_B的二选一控制，用Verilog HDL可以描述为：

```
assign  ALU_B = rs2_imm_s ? imm32 : B;
```

3、I型指令的数据通路与执行过程

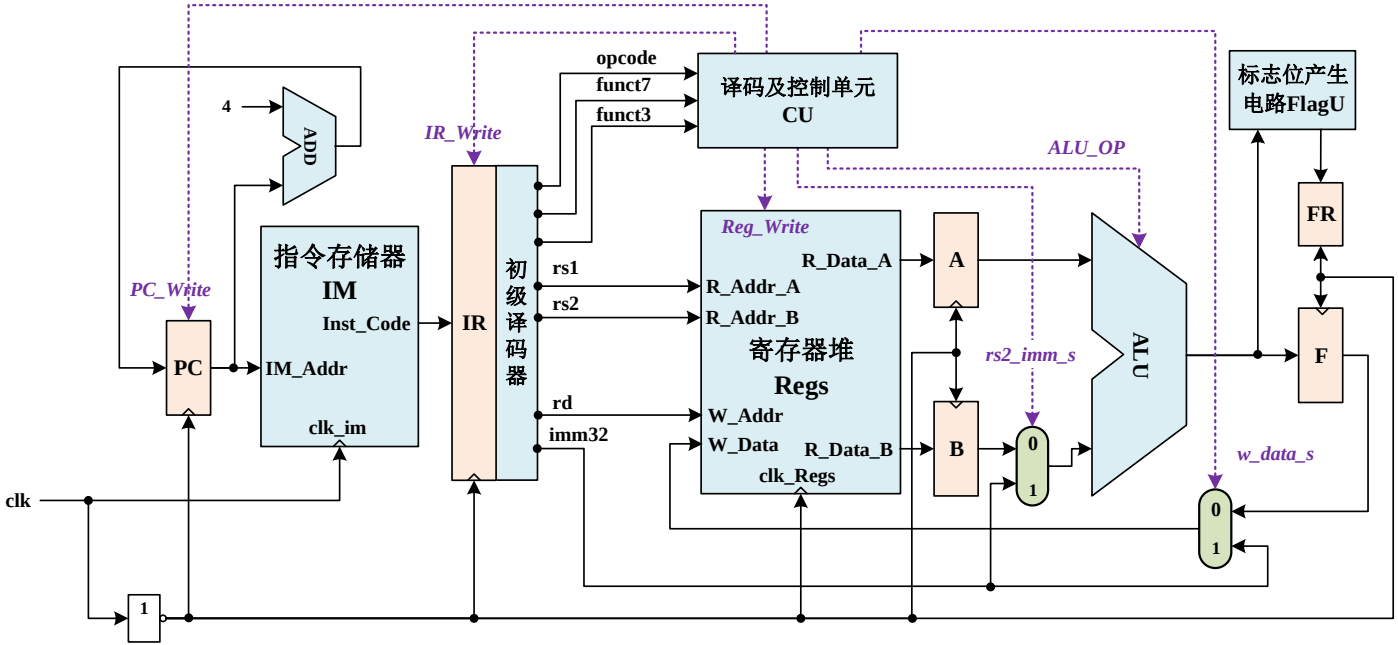
2）I型指令的执行过程

- M2周期：执行A和imm32的运算
- M1周期：rs2寄存器的读取虽然执行，但是无效
  - I型指令中无rs2字段
  - 读出rs2的数据存入B暂存器后，后续并未使用



4、U型指令的数据通路与执行过程

■ 1）U型lui指令的数据通路



4、U型指令的数据通路与执行过程

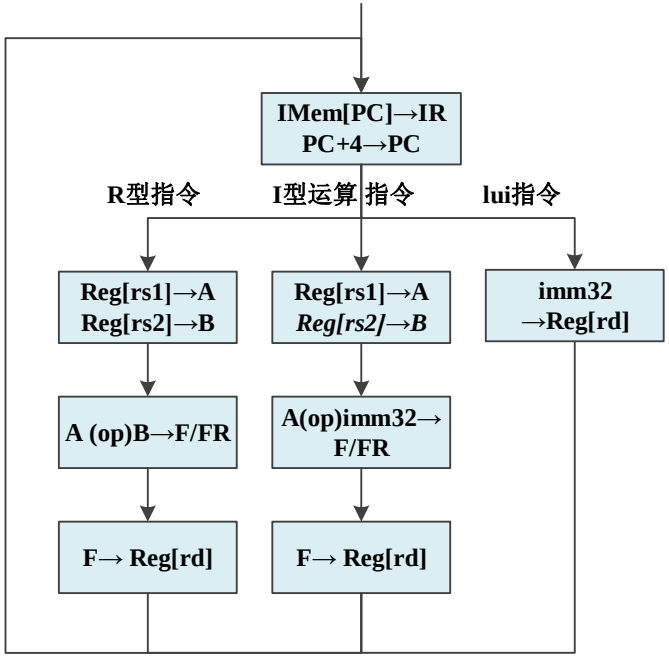
- 1）U型lui指令的数据通路

- 立即数拼接与扩展器ImmU：产生imm32
- 产生的imm32写入寄存器rd：使用一个二选一数据选择器，由w_data_s信号选择
  - w_data_s=0：选择将F写入寄存器堆
  - w_data_s=1：选择将imm32写入寄存器堆

4、U型指令的数据通路与执行过程

2）U型lui指令的执行过程

- lui指令：只需要两个周期
  - 取指令
  - 写回rd
- 每条指令执行完，都回到取指令周期，就是M0周期



计算机的工作过程就是循环往复的取指令、分析指令和执行指令的过程。

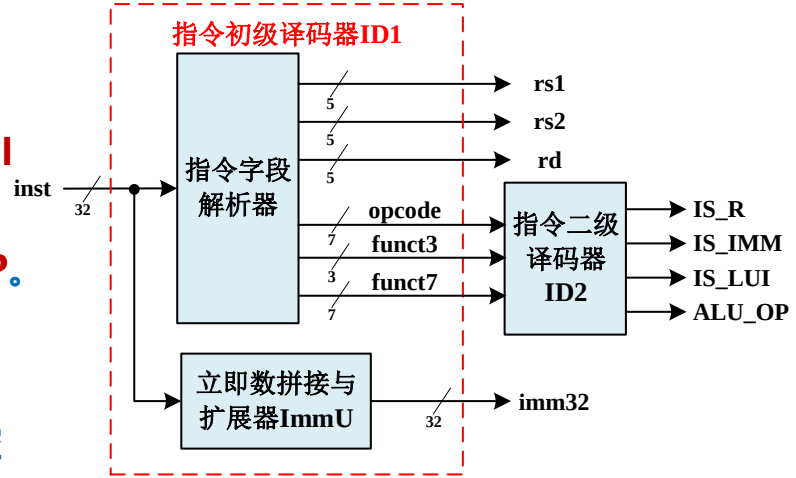
5、指令二级译码

指令的二级译码模块ID2，根据opcode、funct3和funct7三个字段，识别出具体的指令 。

ID2模块译码出本实验的三种类型的指令：

- IS_R、IS_IMM和IS_LUI分别表示当前指令是R型指令、I型运算指令、lui指令；
- ID2还译码产生本条指令需要的ALU运算功能码ALU_OP。

ID2根据opcode字段判定是什么指令；如果是R型指令或者I型运算指令，则再将funct3和funct7字段，翻译成ALU的控制信号ALU_OP，以指定ALU的运算功能。



5、指令二级译码

将实验3中的ALU功能码和R型指令的funct3、funct7做一个匹配。

指令	funct7	funct3	指令运算操作	ALU_OP
add rd,rs1,rs2	0000000	000	算术加	0000
sub rd,rs1,rs2	0100000	000	算术减	1000
sll rd,rs1,rs2	0000000	001	左移	0001
slt rd,rs1,rs2	0000000	010	有符号数小于置位	0010
sltu rd,rs1,rs2	0000000	011	无符号数小于置位	0011
xor rd,rs1,rs2	0000000	100	位异或	0100
srl rd,rs1,rs2	0000000	101	逻辑右移	0101
sra rd,rs1,rs2	0100000	101	算术右移	1101
or rd,rs1,rs2	0000000	110	位或	0110
and rd,rs1,rs2	0000000	111	位与	0111

R型指令的ALU_OP可以直接从funct7的次高位和funct3进行拼接：

```
assign  ALU_OP = {funct7[5],funct3};
```

5、指令二级译码

除了右移指令（funct3=101b），需要借助funct7字段区分逻辑右移和算术右移指令外，其他直接在funct3编码前加上一个0即可。

对于I型运算指令，除了移位指令外，没有funct7字段。

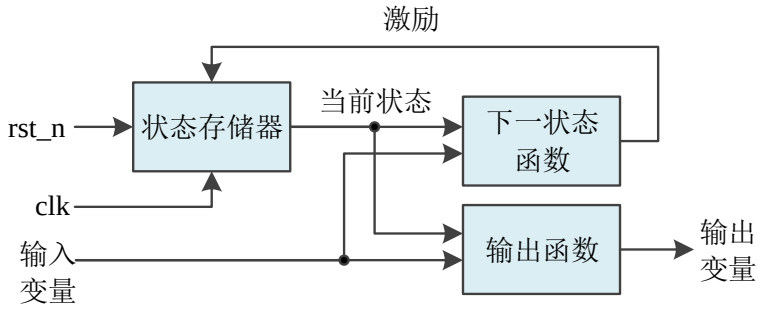
指令	funct7	funct3	指令运算操作	ALU_OP
addi rd,rs1,imm12	—	000	算术加	0000
slli rd,rs1,shamt	0000000	001	左移	0001
slti rd,rs1,imm12	—	010	有符号数小于置位	0010
sltiu rd,rs1,imm12	—	011	无符号数小于置位	0011
xori rd,rs1,imm12	—	100	位异或	0100
srli rd,rs1,shamt	0000000	101	逻辑右移	0101
srai rd,rs1,shamt	0100000	101	算术右移	1101
ori rd,rs1,imm12	—	110	位或	0110
andi rd,rs1,imm12	—	111	位与	0111
addi rd,rs1,imm12	—	000	算术加	0000

Verilog HDL可以描述为：

```
assign ALU_OP = (funct3==3'b101) ? {funct7[5],funct3}:{1'b0,funct3};
```

6、控制单元CU

- 1）有限状态机
  - rst_n：复位信号，用于确定有限状态机的初始状态；
  - clk：时钟信号，用于同步状态转换的边沿。
  - 状态存储器：储存状态机状态的一组触发器，触发器的次态由下一状态函数决定
  - 输出函数：状态机的输出，它是当前状态和输入变量的组合逻辑函数。



使用FPGA设计有限状态机的步骤如下：

（1）确定输入、输出与状态机状态，画出状态转移图；

（2）状态化简，得到最简的状态转移图；

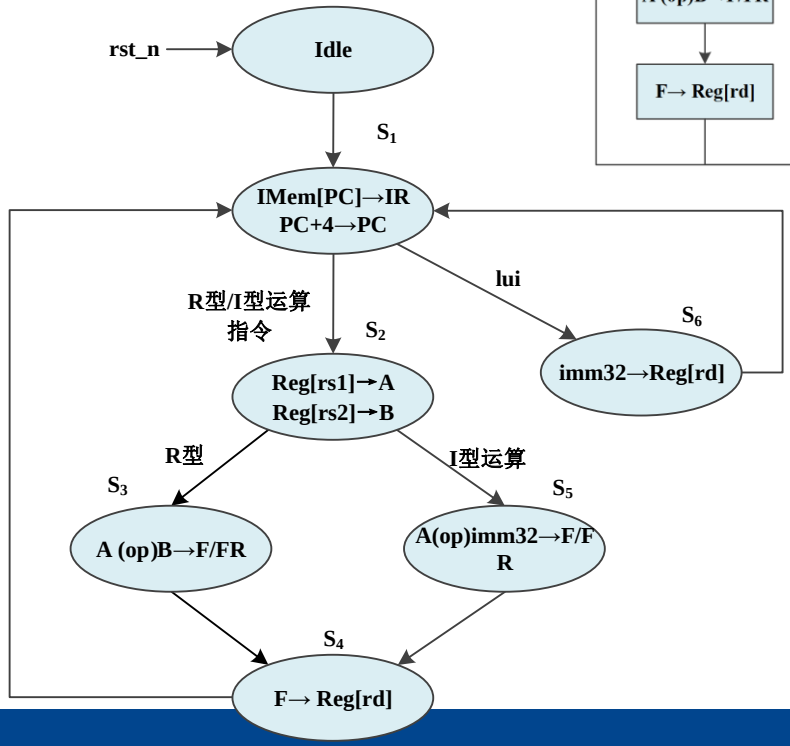
（3）对状态进行编码：有binary、gray、one-hot等，在FPGA开发平台中，可以进行选择和自定义；

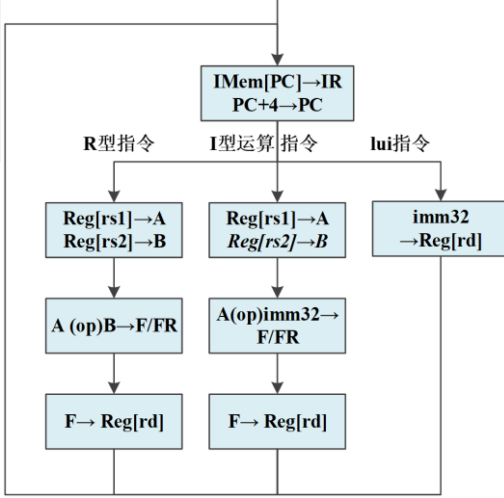
（4）使用HDL描述，并综合、仿真与实现。

6、控制单元CU

2）指令执行的状态转移图

- Idle：空闲状态，是复位信号来临后的第一个状态，也是开机上电后的初态；
- S1状态：取指令状态，它的次态是根据指令译码结果进行分支；
- S2状态：R型指令和I型运算指令的M1周期合并而成的；S2的次态，则按照R型指令和I型运算指令进行分支，分别转移到S3和S5；
- 最后S3和S5都转移到S4，执行写回操作。





6、控制单元CU

3）控制单元CU的有限状态机设计

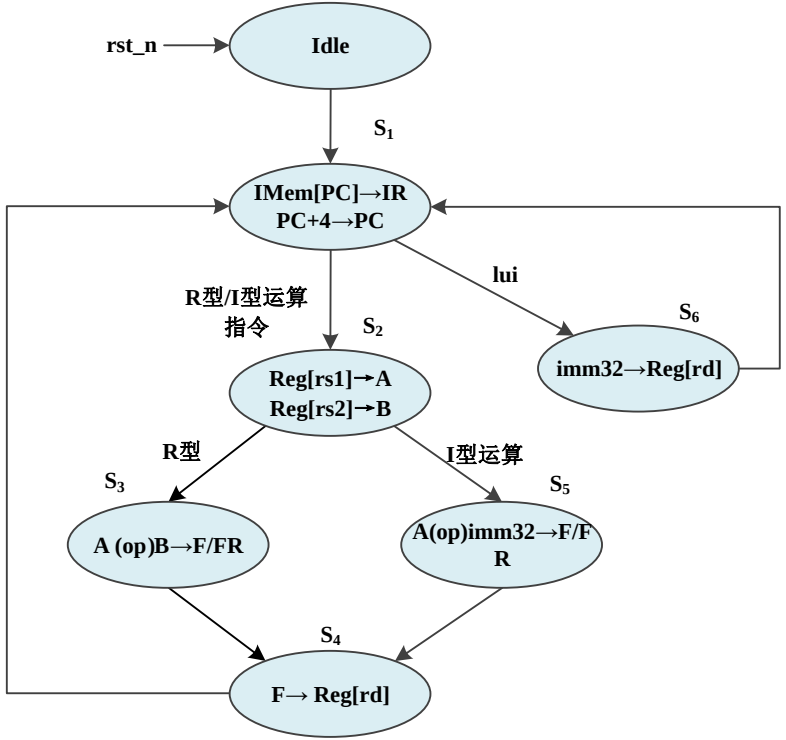
- 三段式方法描述多周期CPU的控制单元，即：状态转移、次态函数和输出函数各用一个独立的always语句实现。

```
// 第一段：状态转移，在clk的边沿进行状态转移，是同步时序逻辑电路
always @(negedge rst_n or posedge clk)
begin
        if (!rst_n)   ST <= Idle;        // 初始状态
        else          ST <= Next_ST;     // clk的上跳沿，进行状态转移
end
```

6、控制单元CU

■ 3）控制单元CU的有限状态机设计

```
// 第二段：次态函数，对次态的阻塞式赋值，是组合逻辑函数
always @(*)
begin
        Next_ST = Idle;
        case (ST)     // 根据状态转移图进行次态赋值
                Idle:  Next_ST = S1;
                S1: begin
                        if (IS_LUI)  Next_ST = S6;
                        else         Next_ST = S2;
                    end
                ......
                default:  Next_ST = S1;
        endcase
end
```



6、控制单元CU

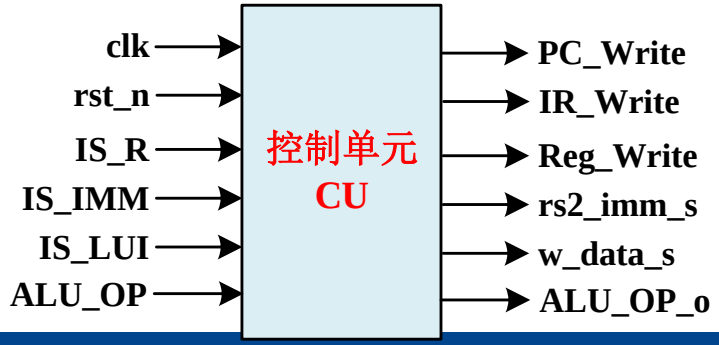
3）控制单元CU的有限状态机设计

```
// 第三段：输出函数，在clk的上跳沿，根据下一状态进行控制信号的非阻塞式赋值
always @(negedge rst_n or posedge clk)
begin
   if (!rst_n)                    // 全部信号初始化为0
      begin
         PC_write <= 1'b0;   IR_write <= 1'b0;
         ......
      end
    else
      begin
         case(Next_ST)
            S1:begin          // 信号的非阻塞式赋值
                  PC_write<=1'b1;  IR_write<=1'b1;.....
               end
            ......                  // 每个状态下输出的控制信号
         endcase
      end;
end
```

4）状态输出的控制信号

控制信号 \ 状态	S_1	S_2	S_3	S_4	S_5	S_6
PC_Write	1	0	0	0	0	0
IR_Write	1	0	0	0	0	0
Reg_Write	0	0	0	1	0	1
ALU_OP[3:0]	—	—	****	—	****	—
rs2_imm_s	—	—	0	—	1	—
w_data_s	—	—	—	0	—	1

抽象出CU模块入输出信号，将ID2模块的输出信号作为输入，然后再根据状态机的不同状态，输出控制信号



7、测试程序

①测试程序翻译成机器指令代码

②通过COE文件，放在指令存储器0号单元开始的地方

```
main:
    addi    x1,x0,-0x78A    #x1=0xFFFF_F876
    addi    x2,x0,4         #x2=0x0000_0004
    add     x3,x1,x2        #x3=0xFFFF_F87A
    sub     x4,x1,x2        #x4=0xFFFF_F872
    sll     x5,x1,x2        #x5=0xFFFF_8760
    srl     x6,x1,x2        #x6=0x0FFF_FF87
    sra     x7,x1,x2        #x7=0xFFFF_FF87
    slt     x8,x1,x2        #x8=0x0000_0001
    sltu    x9,x1,x2        #x9=0x0000_0000
    and     x10,x5,x6       #x10=0x0FFF_8700
    or      x11,x5,x6       #x11=0xFFFF_FFE7
    xor     x12,x5,x6       #x12=0xF000_78E7
    lui     x13,0x80000     #x13=0x8000_0000
    addi    x14,x13,-1      #x14=0x7FFF_FFFF
    addi    x15,x14,0x123   #x15=0x8000_0122
    slli    x16,x15,3       #x16=0x0000_0910
    srli    x17,x15,3       #x17=0x1000_0024
    srai    x18,x15,3       #x18=0xF000_0024
    slti    x19,x18,-1      #x19=0x0000_0001
    sltiu   x20,x18,-1      #x20=0x0000_0001
    slti    x21,x18,1       #x21=0x0000_0001
    sltiu   x22,x18,1       #x22=0x0000_0000
    andi    x23,x12,0xFF    #x23=0x0000_00E7
    ori     x23,x12,0xFF    #x23=0xF000_78FF
    lui     x24,0x00010     #x24=0x0001_0000
    addi    x24,x24,-1      #x24=0x0000_FFFF
    xori    x25,x24,-1      #x25=0xFFFF_0000
```

三、实验要求

1. 新建一个工程，编写二级指令译码模块ID2和控制单元CU。
2. 编写用于测试的汇编程序RIU_test.s，可以自行编写，也可以采用前述测试程序；将其翻译成机器指令代码，写入COE文件。
3. 新建一个ROM类型的64×32位的指令存储器IM，选择上述COE文件初始化其内容。
4. 将前述实验实现的ALU模块、暂存器模块ABF_Latch、寄存器堆模块Regs、取指令模块IF、指令初级译码器模块ID1，与ID2、CU进行连接，构建RIU_CPU模块。
5. 对RIU_CPU模块进行仿真，在rst_n之后，每来一个clk，观察指令执行的每一步骤（状态）是否符合预期，确保RIU_CPU能正确执行目标指令集上的程序。
6. 针对使用的实验板卡，设计RIU_CPU的板级验证实验方案，编写顶层测试模块，要求至少能观察PC、IR、当前所处的状态编码ST、W_Data、标志位。

三、实验要求

7. 复位后，按clk按键，会按步骤执行指令，验证每条指令需要几个周期执行完，执行结果是否正确，将实验结果记录到表中。

序号	PC	IR	汇编指令	预期 执行结果	指令clk数	W_Data

三、实验要求

8. 撰写实验报告，格式见附录，重点内容包括：

- 1）对仿真结果进行**分析**；描述你设计的板级验证实验方案、模块结构与连接；说明你的板级操作过程；记录板级实验结果。
- 2）针对以下有待验证的实验问题，给出证据与分析，得到有效结论：

  ①I型运算指令执行正确；

  ②R型运算指令执行正确；

  ③lui指令执行正确。所谓执行正确，指：能进行正确的运算，并存入规定寄存器；指令执行经过的流程状态符合预期。
- 3）请力所能及回答或实践本实验的“**思考与探索**”部分。

四、实验步骤

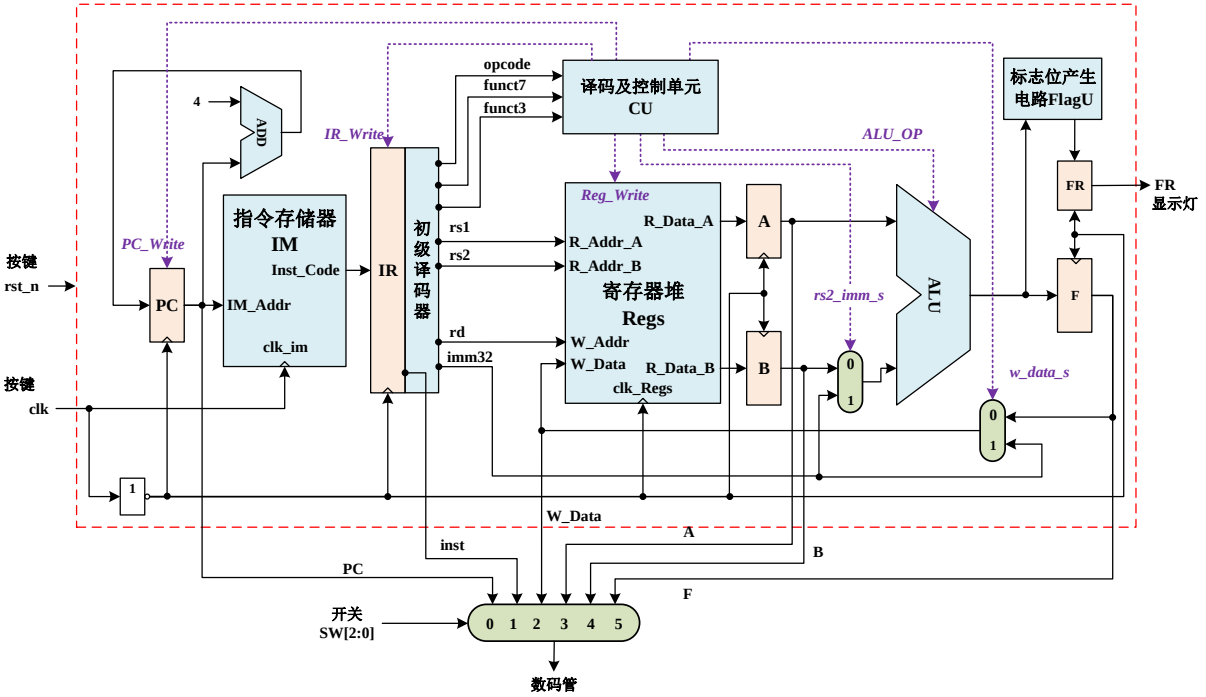
1. 新建一个工程，编写二级指令译码模块ID2；
2. 定义控制单元CU的输入输出端口，然后按照三段式自动状态机编写程序。
3. 编写用于测试的汇编程序RIU_test.s，可以自行编写，也可以采用前述测试程序；
4. 新建一个RIU_test.coe文件，用实验6的汇编与反汇编功能，将测试程序的反汇编出的机器指令代码，逐个写入RIU_test.coe文件。
5. 通过IP核生成向导创建一个64×32位的单端口ROM存储器，作为指令存储器IM，选择RIU_test.coe文件初始化其内容。
6. 将前述实验实现的ALU模块、暂存器模块ABF_Latch、寄存器堆模块Regs、取指令模块IF、指令初级译码器模块ID1拷贝到工程文件中，并加入工程。

四、实验步骤

7. 将拷贝过来的这些模块与ID2、CU模块进行连接，构建成新的RIU_CPU整机模块。注意，此时可以考虑将部分要观察的信号从内部引出到输出端口。
8. 编写激励仿真代码，对RIU_CPU模块进行仿真测试，首先rst_n=0，然后置1，之后，周期性地产生clk，观察输出信号，分析指令执行的每一步骤（状态）是否符合预期。确保RIU_CPU能正确执行目标指令集上的程序。
9. 依据实际板卡情况，设计RIU_CPU模块的板级验证实验方案，然后据此编写一个顶层测试模块；按照要求至少能观察PC、IR、状态编码ST、W_Data及标志位。这需要修改RIU_CPU模块的输出端口，将内部模块的相应信号引出。

四、实验步骤

一个板级验证实验方案：



四、实验步骤

10. 新建管脚约束文件，进行相应的引脚配置。
11. 生成*.bit文件，下载到实验设备的FPGA芯片中。
12. 板级实验：按照你所设计的实验方案，操作输入设备、观察输出设备，预期的验证操作如下：
    - 1）按rst_n按键，将PC和IR清零，接下来将从0号单元开始执行指令；
    - 2）按动时钟键clk，每按一下，指令就执行一步，观察PC和IR的值，判断每条指令需要几个clk才能执行完；观察W_Data，判断指令执行结果是否正确。

五、思考与探索（至少完成1道）

1. 仿真测试中，你的PC和IR始终保持一致吗？如果不一样，分析原因。
2. 板级验证中，你的PC和IR始终保持一致吗？如果不一样，分析原因。
3. 板级验证中，你每按动一次clk，是否就执行一步操作？状态如果不是，分析原因，尝试解决。
4. 目前的设计是：每按动一次clk，就执行指令中的一步操作，一条指令需要按动2次clk或者4次clk才能完成；如果想要实现：每按动一次clk，就执行一条指令，请思考如何实现，并进行尝试。
5. 本次实验实现的U型指令是lui指令，还有一条auipc指令未实现：auipc rd,imm20

   # PC+{imm20,12{0}}→rd，仿照前述数据通路的设计方法，实现auipc指令，并修改你的程序，完成设计。

五、思考与探索（至少完成1道）

6. 对于本实验的目标指令集，PC_Write和IR_Write必须要有吗?
7. 如果按照硬布线控制器的设计方法，那么CU模块完全不同，它应该是完全的组合逻辑电路，但是此时需要设计一个时序系统自动产生M0~M3。感兴趣的读者，可尝试用硬布线控制器的设计方法，完成本实验的目标指令集。
8. 说说你在实验中碰到了哪些问题，你是如何解决的?